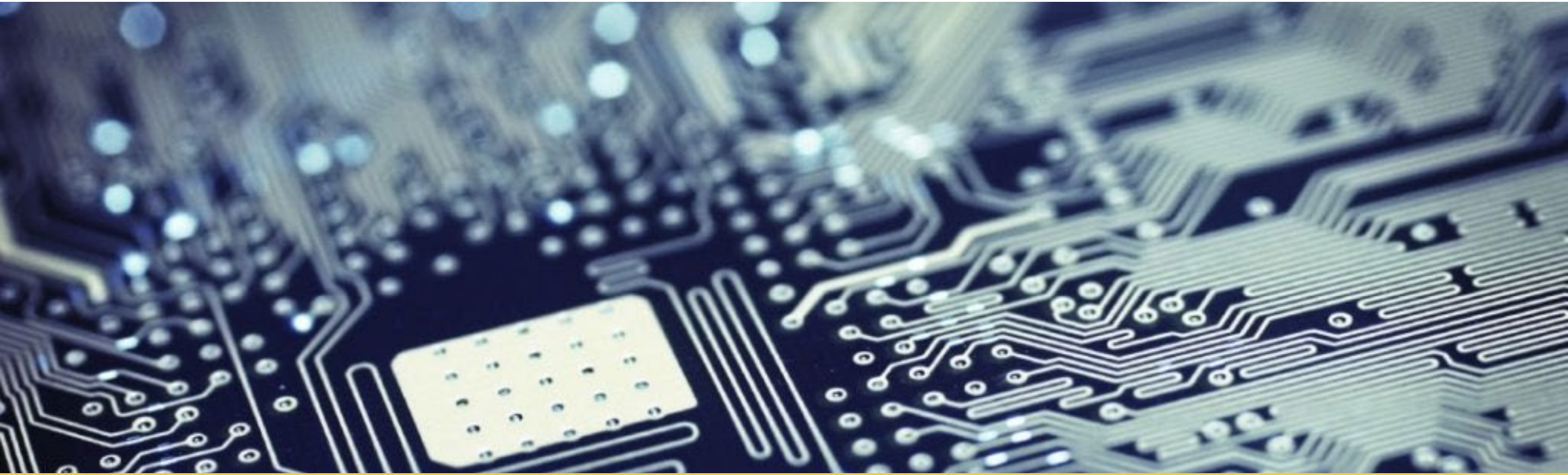
2BIT
IT's our passion.

Angular Architekturen

Von MVC bis Redux



MVC

2BIT
IT's our passion.

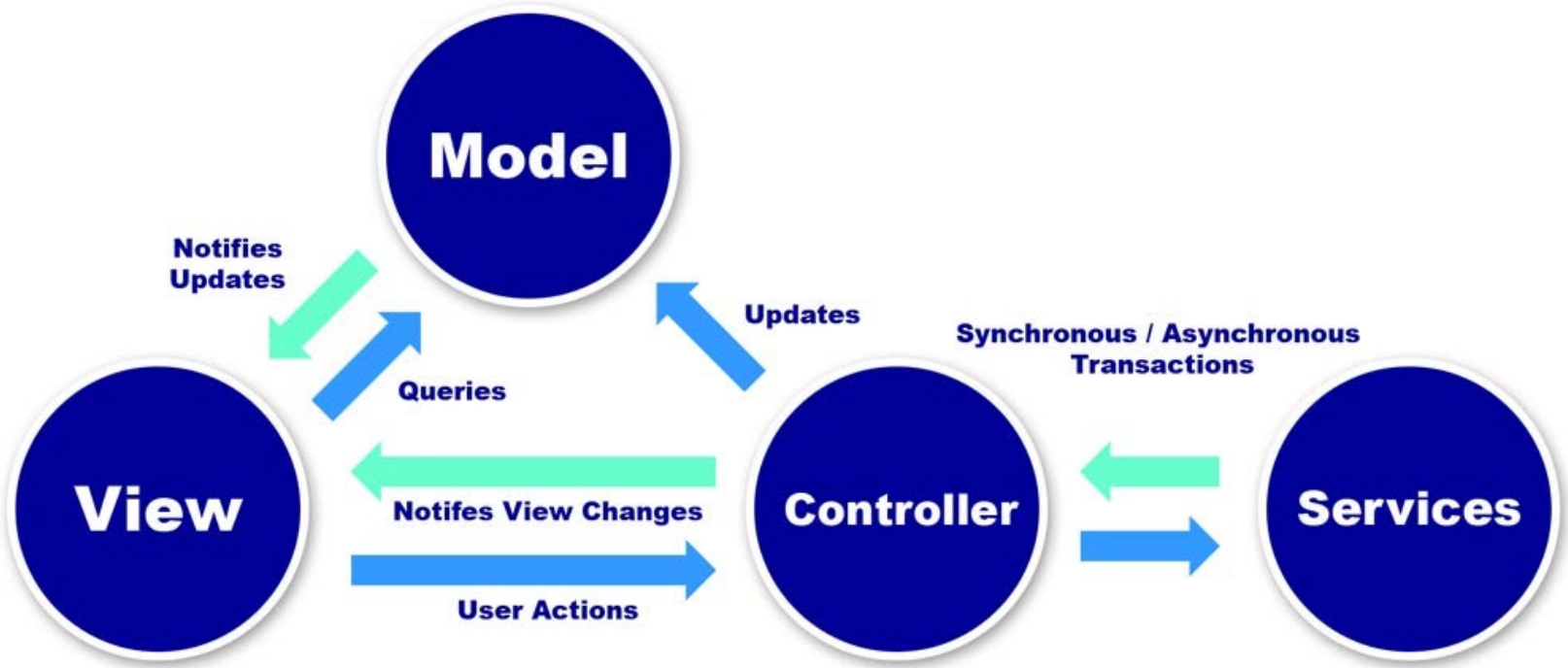
- MVC (Model-View-Controller)
  - Model
    Data Model for the View
  - View
    Displays the data Model
  - Controller
    Handles User Input
    Handles Business Logic
- Nachteil einer klassischen MVC Architektur ist, dass sich im Kontroller sehr viel Logik ansammelt, welche zusätzlich noch sehr schwierig zu testen ist

MVC

2BIT
IT's our passion.

- Vorteile
  - Klares Pattern welches bereits in unzähligen UI Frameworks Verwendung findet
  - Klare Separierung zwischen den verschiedenen Aufgaben welche in einem UI erledigt werden müssen
- Nachteile
  - Das Pattern lässt sehr viele Freiheiten wie der Code gestaltet werden soll (Anzahl Klassen, Klassenseparierung etc.)
  - Es ist nicht klar definiert wo die Client Business Logik abgelegt werden soll
  - Tendenz den Controller mit Business Logik zu überladen

MVC + S


Model
Notifies
Updates
Queries
Updates
Synchronous / Asynchronous
Transactions
View
Notifes View Changes
User Actions
Controller
Services

MVC + S

2BIT
IT's our passion.

- Vorteile gegenüber klassischem MVC
  - Es wird eine Komponente hinzugefügt welche für die Business Logik verantwortlich ist. Dadurch kann einer sehr hohen Ansammlung von Code im Controller entgegen gewirkt werden
  - Services sind wiederverwendbar und so kann einfach Business Logik zwischen verschiedenen Controllern geteilt werden
  - Die gesamte Ladelogik von Daten wird dadurch ebenfalls vom Controller in Services ausgelagert

MVC + S Übung

- Ladet euch die Ausgangslage herunter und versucht die klassische MVC Architektur in eine MVC + S Architektur umzuwandeln
  - extrahieren von Logik aus der Komponenten in einen Service

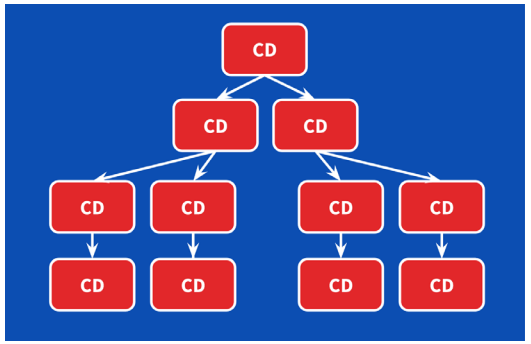
- Ausgangslage
  - Lösung von Component Communication Übung
  - https://github.com/rdnscr/2school-angular-components

MVC + S Übung

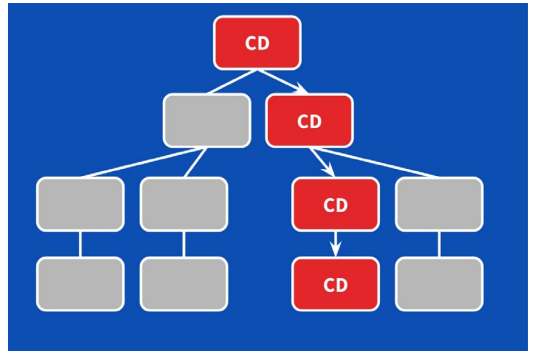
2BIT
IT's our passion.

- Schritte zur Lösungserarbeitung
  - Einen neuen Service im noch fehlenden «service» Ordner erstellen
  - Den Service mit dem entsprechenden Code ausfüllen und im Module registrieren
  - Der Service soll eine Funktion zum Laden der Daten aus dem JSON File anbieten
    - Wir wollen gerade ein Observable mit der Liste von TodoItems zurück geben und der Konsument des Services muss sich nicht um das zurück mappen des JSONs auf JavaScript Objekte kümmern
  - Neu soll im ngOnInit der Service zum Laden der initialen Daten verwendet werden

ChangeDetectionStrategy - OnPush



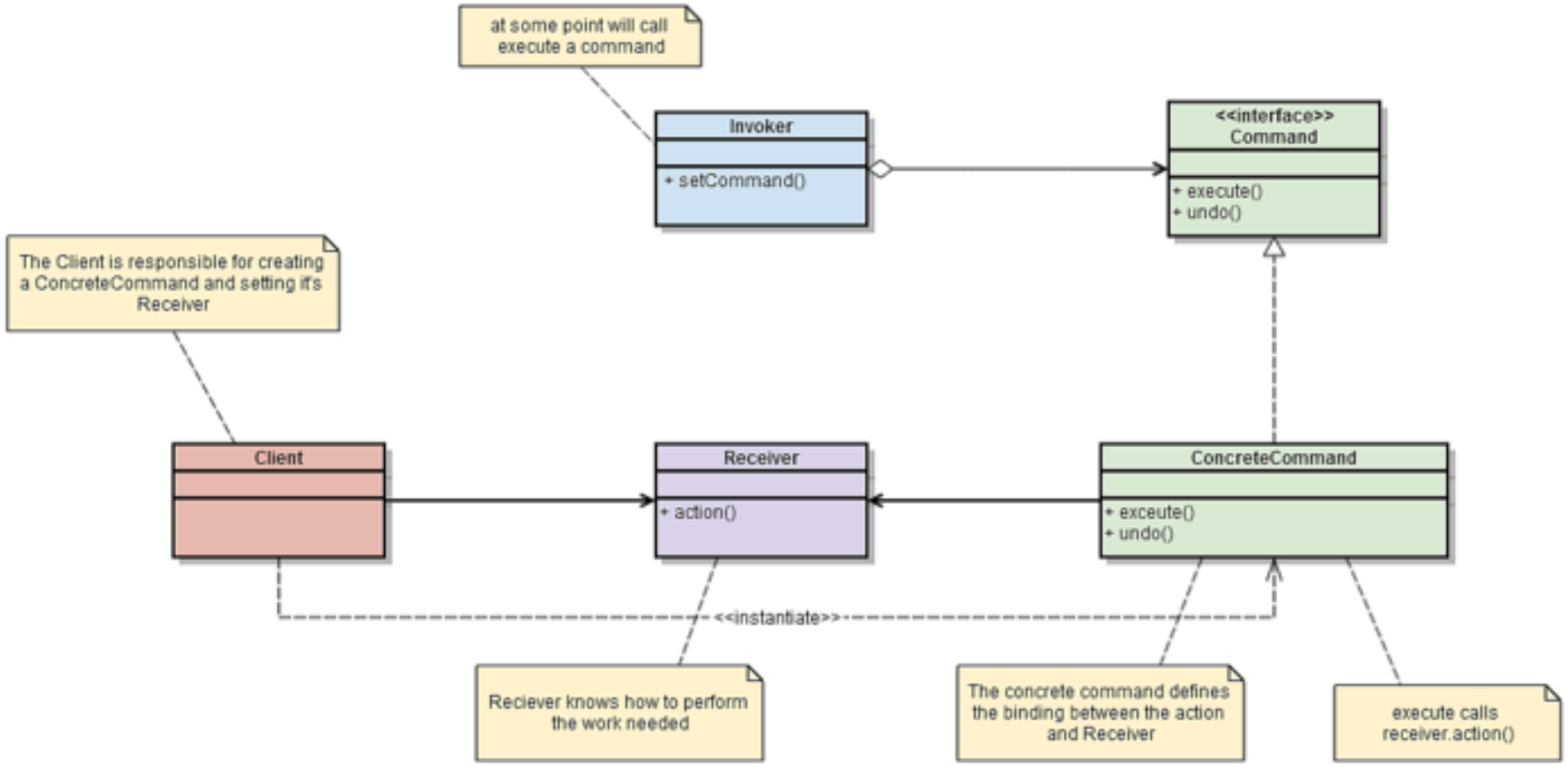
- Immutable State
  - Nicht veränderbar
- Angular ChangeDetectionStrategy Default
  - Hört auf alle Änderungen auch innerhalb des Objekts
- Angular ChangeDetection OnPush
  - Hört nur auf Änderungen wenn das komplette Objekt ändert



- Beispiel => Todo OnPush
- Probiert es mal aus!

Command Pattern

The Command Pattern encapsulates a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations.

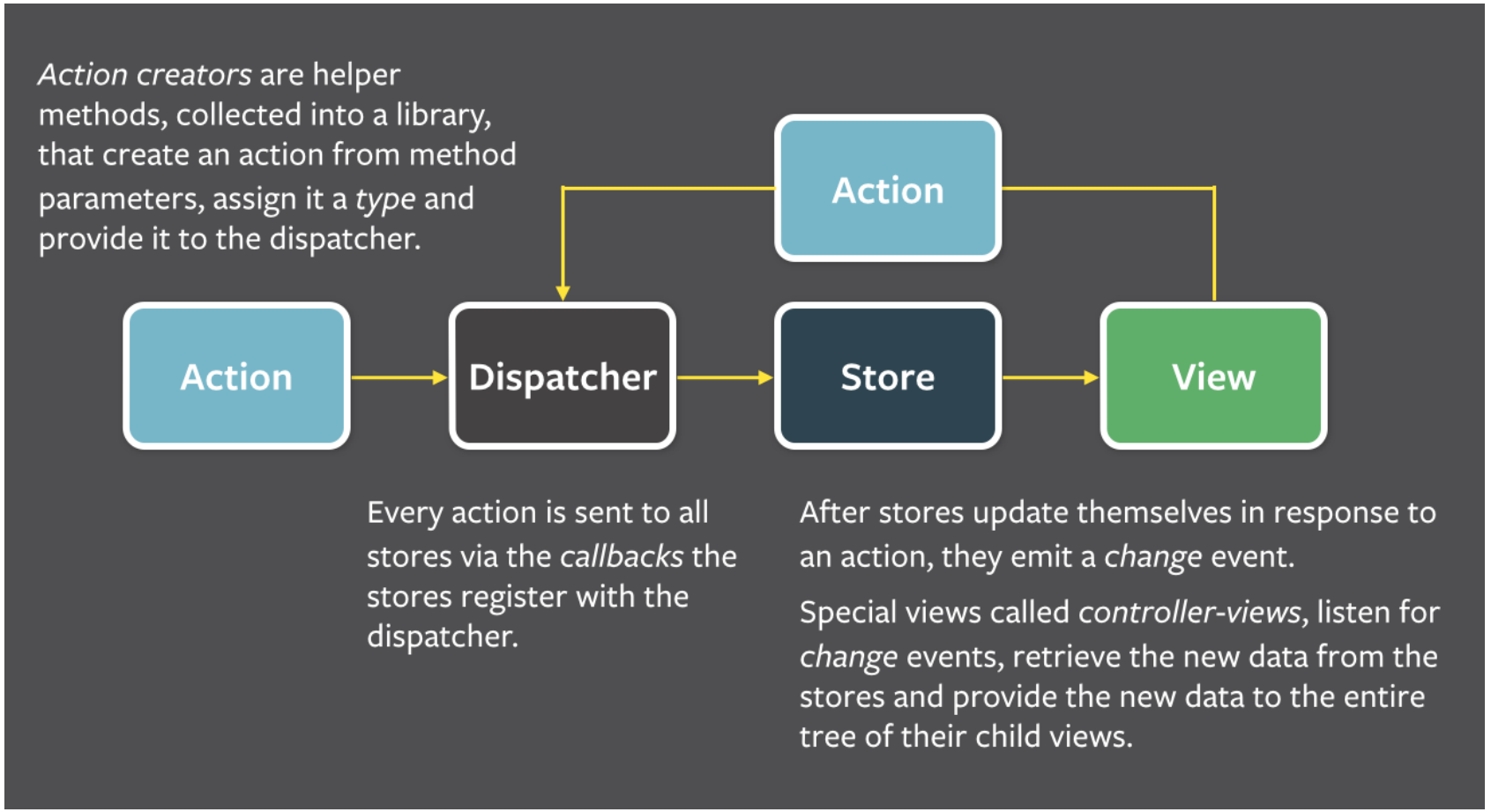


Flux

2BIT
IT's our passion.

- Flux is the **application architecture that Facebook uses** for **building client-side web applications**. It complements React's composable view components by utilizing a **unidirectional data flow**. It's more of **a pattern rather** than a formal framework, and you can start using Flux immediately without a lot of new code.
- https://facebook.github.io/flux/

Flux – single store



Flux – single store

2BIT
IT's our passion.

- Action
  - Information was für eine Aktion der Store ausführen soll
- Dispatcher
  - Message Bus über den die Aktionen von überall ausgelöst und über den Bus an den oder die Stores geschickt werden können
- Store
  - Repräsentiert sowohl die Datenhaltung einer Domäne oder Applikation als auch die dazugehörige Businesslogik
- View
  - Kümmert sich um die Darstellung der Daten

Flux – single store

2BIT
IT's our passion.

- Vorteile
  - Weitere Entkopplung zwischen Controller (Component) und der Businesslogik
  - Daten sind in einem zentralen Store gehalten und können so einfach über verschiedene Views geteilt werden
- Nachteile
  - In der «single store» Variante ist die Wahrscheinlichkeit sehr hoch, dass der Store sehr komplex und unübersichtlich wird

Flux – multi store

- Der Vorteil gegenüber der Flux Variante mit nur einem Store ist, dass die Business Logik auf die verschiedenen Stores aufgeteilt werden kann
- Es kann pro Domäne ein eigener Store erstellt werden, welcher als «Information Expert» für diese Domäne dient
- Es ist schwierig Store übergreifend Informationen / Daten auszutauschen
- Die Zuweisung der Verantwortlichkeiten eines Stores ist nicht ganz einfach ohne eine Store Inflation zu erreichen

Flux – single store

- Ladet euch die Ausgangslage herunter und versucht die klassische MVC Architektur in eine Flux Single Store Architektur umzuwandeln

https://github.com/rdnscr/2school-angular-advanced

Flux – single store

2BIT
IT's our passion.

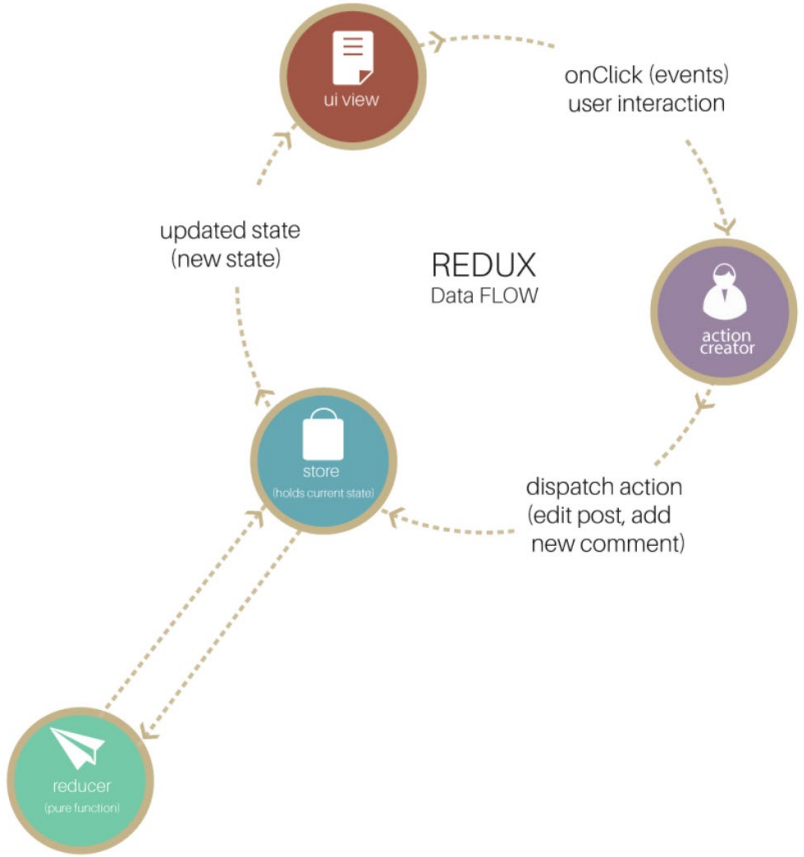
- Schritte zur Lösungserarbeitung (1/2)
  - Neuen Ordner «services» erstellen
  - Neues File anlegen für die Actions die es gibt
    - Enum mit den Action Typen Add, Load, Check, Uncheck, Reset
    - Action Klasse welche als Konstruktorparameter den Action Typ, eine optionale Id und einen optionalen string als Description enthält
  - Ein neues TypeScript File erstellen für die Konfiguration des Flux Stores im Injection System
    - Ein InjectToken für den fluxDispatcher erstellen
    - Eine Konstante erstellen mit der entsprechenden Konfiguraiton für das Injection System
  - Registrieren der Flux Konfiguration in Module

Flux – single store

2BIT
IT's our passion.

- Schritte zur Lösungserarbeitung (2/2)
  - Einen neuen Service schreiben um die Kommandos welche über den Dispatcher gesendet werden abzuarbeiten
    - Der Service beinhaltet auch die Datenhaltung in Form von zwei Arrays. Einen für den Ursprungszustand (benötigt für die Reset Funktion) und einen für den aktuellen (manipulierten) Zustand
    - Der neue Service muss den dispatcher der Actions als Konstruktorparameter haben und sich anschliessend darauf subscriben (Action Stream)
    - Anschliessend muss in der Subscribe Funktion die jeweilige Action ausgewertet und der entsprechende Manipulationscode ausgeführt werden
    - Die Logik der Applikation soll sich nicht verändern! Sie wird lediglich in den Service verschoben und durch das feuern von Actions getriggert
  - Anpassen der verschiedenen Komponenten um Actions auszulösen

Redux


ui view
onClick (events)
user interaction
updated state
(new state)
REDUX
Data FLOW
action
creator
store
(holds current state)
dispatch action
(edit post, add
new comment)
reducer
(pure function)

Redux

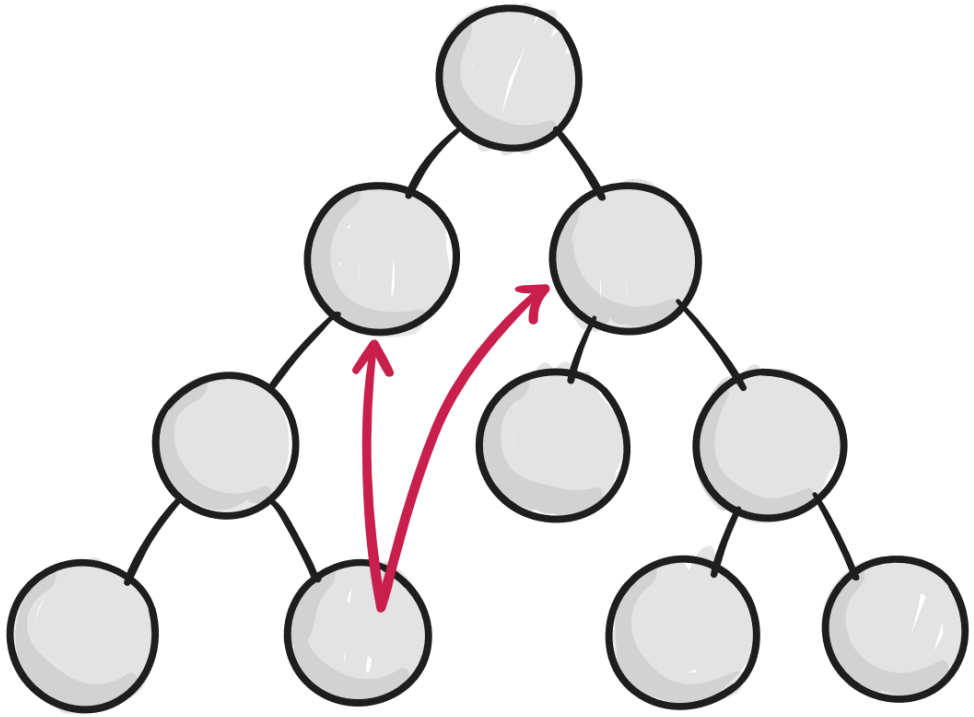
2BIT
IT's our passion.

- Store
  - Immutable
    - Jede Änderung bewirkt, dass ein neuer State erstellt wird
    - Dadurch können auch die Replay Tools welche es für Redux für verschiedene Browser gibt verwendet werden
  - Der Store wiederspiegelt denn aktuellen Applikationszustand und kann auch verhängt werden mit dem Router State falls gewünscht
- Actions
  - Beschreibt die Aktion welche auf dem Store ausgeführt werden soll

Redux

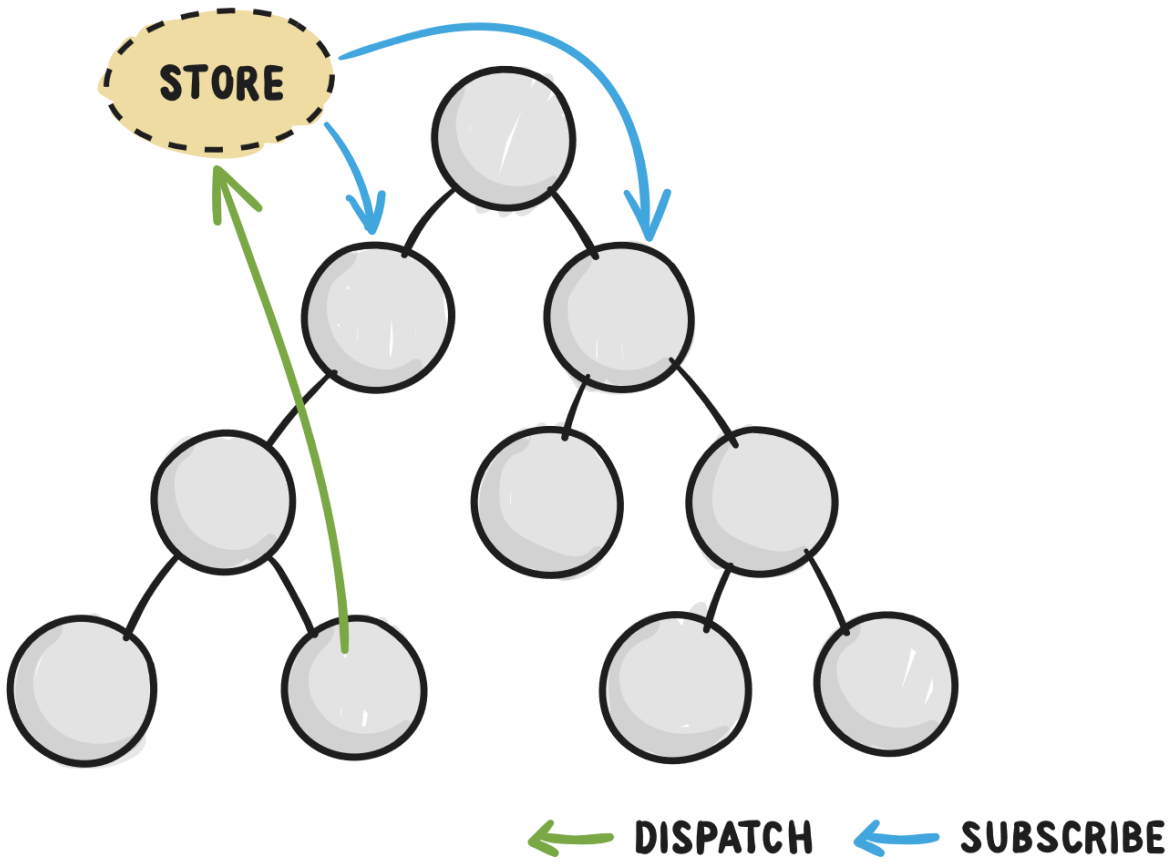
- Reducers
  - Führen durch eine Action angestossene Businesslogik effektiv auf einem neuen State aus
  - Sind «pure functions» und können dadurch sehr einfach getestet werden
- Effects
  - Führen durch eine Action angestossene Businesslogik aus, welche nicht als «pure functions» implementiert werden können (bspw. das Laden von Daten)
  - Lösen anschliessend eine weitere Action auf dem Store aus und führen nicht zu einem neuen State innerhalb des Stores

Redux

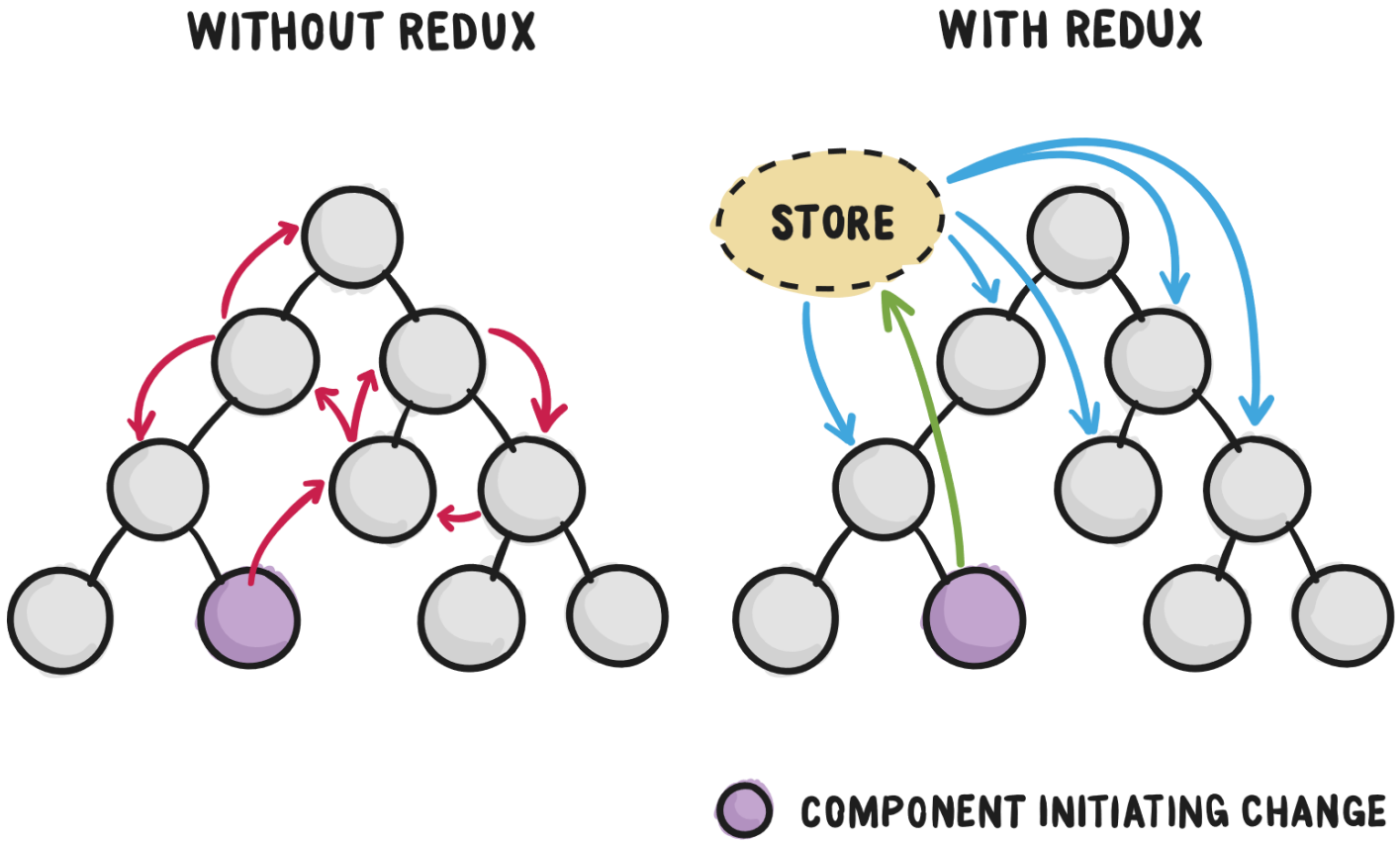


POOR PRACTICE WHEN COMPONENTS TRY TO COMMUNICATE DIRECTLY

Redux



Redux

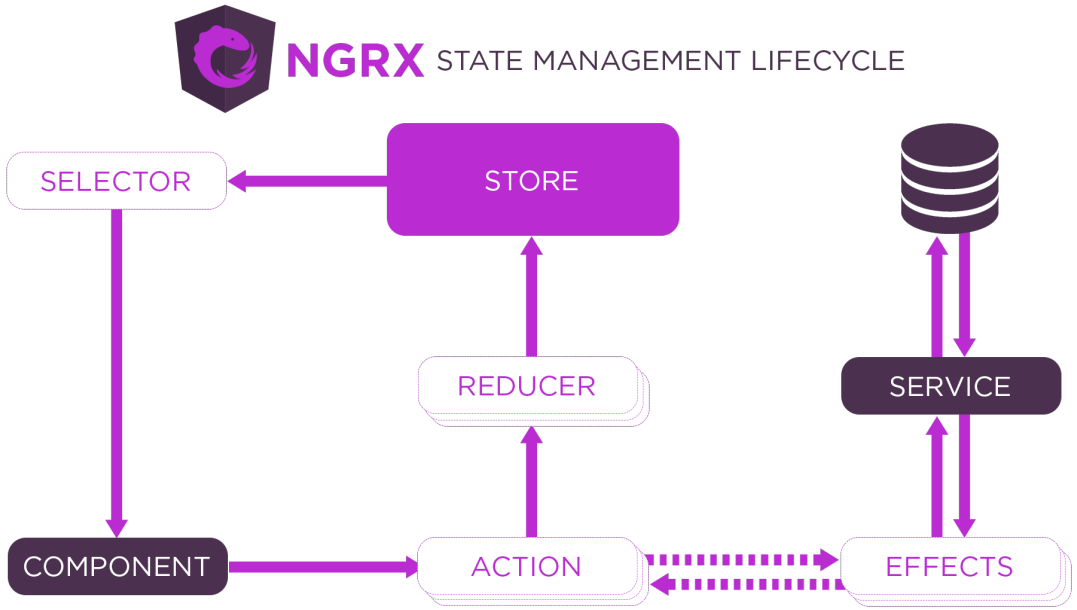

WITHOUT REDUX
WITH REDUX
STORE
COMPONENT INITIATING CHANGE

Redux

2BIT
IT's our passion.

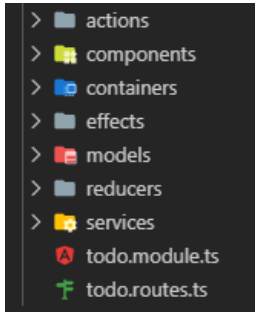
- Vorteile
  - Unidirektionaler Flow von den Controllern (Componenten) über Actions zum Store und dann wiederum über einen neuen Zustand im Store zurück zum Controller (Component)
  - Einfache Testbarkeit der Businesslogik da «pure functions»
  - Controller (Componenten) bleiben sehr leichtgewichtig
- Nachteile
  - Es entsteht eine neue Abhängigkeit zu einer 3rd Party (ngrx)
  - Die gesamte Client Architektur wird komplexer

NgRx



- Action
  - Unique Events in App
  - https://ngrx.io/guide/store/actions
- Reducer
  - Responsible state transitions based on action
  - https://ngrx.io/guide/store/reducers
- Selector
  - Selectors are pure functions used for obtaining slices of store state
  - https://ngrx.io/guide/store/selectors
- Effects
  - Actions to reduce state based on external interactions e.g. network requests and time-based events.
  - https://ngrx.io/guide/effects

NgRx – Folder Structure



- Actions
  - Action definitions of the module. Actions can be defined in multiple files clustered by topics.
- Components
  - A component is responsible of the graphical representation of data. A component only consists of Inputs and Outputs and interacts with its container through these bindings / events.
- Containers
  - Containers read required data from the state with selectors and propagate them to the components. They react onto events from their components and trigger subsequent actions. These actions will invoke the registered effects or reducers.
- Effects
  - An effects invokes asynchronous operations and usually triggers a new action which will invoke a reducer to do some state transformations
- Models
  - Sometimes the Data Transfer Objects don’t fully match the required data for the graphical information. In that case an UI model as interface is defined which makes the representation easier.
- Reducers
  - The reducers change the global state in an immutable way. Reducers can be multiple files usually clustered by topics. In this folder we have in addition the state definition of the module and all the selectors used.
- Services
  - Regular angular service which provides functionality or business logic in a reusable way

Redux

- Weiterführende Informationen zu Redux gibt es hier
  - http://redux.js.org/docs/introduction/
  - https://github.com/ngrx/

- Analysiert das Beispiel unserer Todo Applikation mit Redux

Redux

- Wo befindet sich die Business Logik in unserem Beispiel?
- Wo werden Actions ausgelöst?
- Wo wird der TodoService um die Daten zu laden verwendet und wie wird das Laden der Initialen Daten ausgelöst?
- Was ist der Unterschied zur Flux Single Store Variante?

Redux

- Folgende Ausgangslage befindet sich auf github
  - https://github.com/rdnscr/2school-ngrx
- Implementiert nun das hinzufügen eines neuen Todo Items mit ngrx
  - Hierfür sind im Code verschiedene Todos markiert die gelöst werden müssen
- Lösungsweg
  - Erstellen einer neuen Action um ein Todo hinzufügen
  - Auslösen der Action im Container nachdem die «todo-add» Komponente den entsprechenden Event geworfen hat
  - Erstellen eines reducers welche den Applikationszustand entsprechend anpasst

Architekturentscheidung